# WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Wednesday, August 25, 2004

| Hide? | Set Name | Query | Hit Count |
|---|---|---|---|
| | *DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR* | | |
| ☐ | L3 | 20010209) | 58 |
| ☐ | L2 | ((type adj object) and marshalling) | 85 |
| ☐ | L1 | ((type adj object)) | 15346 |

END OF SEARCH HISTORY

Previous Doc Next Doc Go to Doc#
First Hit

Generate Collection

L3: Entry 11 of 58 File: PGPB Apr 25, 2002

DOCUMENT-IDENTIFIER: US 20020049603 A1
TITLE: Method and apparatus for a business applications server

Application Filing Date:
20010112

Detail Description Paragraph:
[0229] In the preferred embodiment the meta-data store contains the definition of each type of object in the system, its attributes, and some basic properties of those attributes. Further, for each type of object, it contains a reference to the methods to invoke, to insert, update, delete or fetch a given instance of that object from the persistent store.

Detail Description Paragraph:
[0249] Every SabaObject has an unchangeable, unique identifier that identifies that particular object in the persistence store. The uniqueness of this identifier is guaranteed across the entire persistence store regardless of the type of object.

Detail Description Paragraph:
[0277] The SabaObject implementation then simply picks up these fields during its normal marshalling and unmarshalling of arguments. Further, the SabaObject also performs the basic checks for nullity as it would normally do.

Detail Description Paragraph:
[0389] In the alternative embodiment, to develop an application using the BDK, an object model of the application domain should be first developed, retaining a separation between objects that represent business processes and those that represent business data. The two types of objects, obviously, map to session beans and entity beans in EJB parlance. A controller object, for instance, would indicate a session bean whereas an object that persists its data would indicate an entity bean. An application would typically also include UI components (such as JSP pages or servlets) which would use such business components. Thus, there are two primary roles from an application development standpoint:

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L3: Entry 14 of 58 File: USPT Jul 20, 2004

DOCUMENT-IDENTIFIER: US 6766350 B1
TITLE: Shared management of data objects in a communication network

Abstract Text (1):
A method of marshalling on a computer communication network makes it possible to marshall a data object from a programming language used by a data processing application to a communication language which can be used by a communication protocol of the computer communication network. This marshalling method comprises the following steps: reading (E1) a data field included in the data object; substituting (E5-E7), for the data field, a computer address associated with the data field when the structure of the data field is complex; and storing (E5-E7) said computer address associated with the data field in a table of associations (T).

Application Filing Date (1):
20000621

Brief Summary Text (2):
It concerns, according to a first aspect, a method of marshalling a data object on a computer communication network

Brief Summary Text (6):
Correlatively, the present invention concerns a device for marshalling a data object, a device for transferring a data object and a device for the updating remote of a data object on a site in a computer communication network, all adapted to implement the aforementioned methods in accordance with the invention.

Brief Summary Text (19):
When the data object is received by another site, a reverse marshalling operation must be applied to the object in order to obtain a representation of the object in the data processing application used on this second site.

Brief Summary Text (22):
However, this WDDX system is largely unsuited to the marshalling and transfer of a data object having data fields with a complex structure. Such a complex structure is observed in particular when a data field points to or contains at least one other data field which itself may be complex.

Brief Summary Text (24):
This way of proceeding considerably affects the performance of the marshalling and transfer of data objects on a network.

Brief Summary Text (32):
To this end, the present invention relates first of all to a method of marshalling a data object on a computer communication network, from a programming language used by a data processing application to a communication language which can be used by a communication protocol of the computer communication network, comprising the following steps: reading a data field included in said data object; substituting, for said data field, a computer address associated with said data field when the

structure of said data field is complex; and storing said computer address associated with said data field in a table of associations.

Brief Summary Text (33):
Thus the method of marshalling a data object according to the invention makes it possible to temporarily replace at least one complex data field with a computer address and if necessary to proceed with the marshalling of this data field to a communication language only when it is required by a site in the communication network.

Brief Summary Text (39):
Thus it is possible to generate in advance representations of the different data fields by means of a computer address stored in the table of associations without actually having to perform the marshalling of this data field from the programming language to the communication language.

Brief Summary Text (40):
This marshalling in the communication language can be postponed until an object is requested by one of the sites in the network via a computer address.

Brief Summary Text (42):
Thus, when the complex data fields are themselves data objects, the marshalling method according to the invention makes it possible to obtain a representation of a data object by means of links pointing to other data objects, similar to the pointers included in the HTML annotation language used on the Internet.

Brief Summary Text (44):
According to an advantageous characteristic of the invention, the marshalling method includes a step of comparing the structure of a data field with a pre-established list of complex data structures.

Brief Summary Text (46):
According to a second aspect of the invention, a method of transferring a data object on a computer communication network comprises the following steps: receiving a computer request to transfer; extracting a computer address from said computer request; identifying a data object associated with said computer address in a table of associations; marshalling said identified data object to a communication language which can be used by a communication protocol of said communication network; and transferring said marshalled data object.

Brief Summary Text (52):
Preferably the remote updating method also includes a step of marshalling the extracted data object to a programming language used by a data processing application of said site.

Brief Summary Text (88):
Correlatively, the present invention concerns a device for marshalling a data object on a computer communication network, from a programming language used by a data processing application to a communication language which can be used by a communication protocol of said computer communication network, having: means of reading a data field included in said data object; means of substituting, for said data field, a computer address associated with said data field when the structure of said data field is complex; and means of storing said computer address associated with said data field in a table of associations.

Brief Summary Text (89):
According to the second aspect of the invention, a device for transferring a data object on a computer communication network has: means of receiving a computer request for transfer; means of extracting a computer address from said computer request; means of identifying a data object associated with said computer address

in a table of associations; means of marshalling said identified data object to a communication language which can be used by a communication protocol of said communication network; and means of transferring said marshalled data object.

Brief Summary Text (97):
The present invention also concerns a computer comprising a device for marshalling a data object and/or a device for transferring a data object and/or a device for remote updating, and/or a device for remote execution, and/or a device for activation of a function on a local computer and/or an interface transfer device, and/or a device for producing an activation computer request and/or a device for activating a function on a distant computer, all in accordance with the invention.

Brief Summary Text (98):
It also relates to a computer communication network including a device for marshalling a data object and/or a device for transferring a data object and/or a remote updating device and/or a remote execution device and/or a device for activating a function on a local computer and/or an interface transfer device and/or a device for producing an activation computer request and/or a device for activating a function on a distant computer, all in accordance with the invention.

Brief Summary Text (100):
The present invention also relates to a computer program stored on a storage means or an information carrier, possibly removable, incorporated or not into a computer, comprising portions of software code or program instructions adapted to implement the steps of the marshalling method according to the invention and/or the steps of the transfer method according to the invention and/or the steps of the remote updating method according to the invention and/or the steps of the remote execution method and/or the steps of the method of activating a function on a local computer and/or the steps of the interface transfer method and/or the steps of the method of producing an activation computer request and/or the steps of the method of activating a function on a distant computer, all in accordance with the invention, when said computer program is run into a computer.

Drawing Description Text (5):
FIG. 4 is a diagram illustrating the direct and reverse marshalling of a data object, from a programming language to a communication language;

Drawing Description Text (8):
FIG. 7 is an algorithm illustrating a method of marshalling a data object according to a first aspect of the invention;

Detailed Description Text (32):
The computer C1 has a marshalling device 10 as depicted schematically in FIG. 4.

Detailed Description Text (33):
Naturally, each computer C1 to C6 in the communication network can include such a marshalling device.

Detailed Description Text (35):
The same computer also has a reverse marshalling device 13 which makes it possible to effect the reverse conversion in order to transform an object 12 depicted in a communication language into an object 11 in a data processing language.

Detailed Description Text (36):
The marshalling operation performed by the marshaller 10 thus makes it possible, on a computer, to make visible the objects created by an application of this computer, or in other words to publish them on the network.

Detailed Description Text (40):
This enables several objects to be sent to a distant application. This distant

application has no need to wait for all the objects to have been received in order to begin their reverse marshalling in the data processing language used by the distant application, by the marshaller 13.

Detailed Description Text (44):
In accordance with the marshalling method of the invention, although the data objects and container objects can be included directly in the data objects marshalled to this communication language, it is preferable that only the literal objects be marshalled and that the other objects be included only by reference.

Detailed Description Text (46):
In accordance with one aspect of the invention, it makes it possible to send several interfaces to distant applications. As before, the distant application has no need to await the reception of all the interfaces in order to effect their reverse marshalling to the language used by the application and to use them.

Detailed Description Text (55):
Preferably the data objects and container objects are replaced at the time of marshalling by references to these objects using a URI computer address.

Detailed Description Text (59):
This corresponds to the generic concept of "function" or "method". A function is identified by its signature, for example a name, the type of input argument used and the type of object obtained when this function is executed.

Detailed Description Text (115):
As illustrated in FIG. 5, the computer C3 has a third application 26 which also uses internal data 23 and external data 24. These external data 24 are also obtained by marshalling certain internal data 23 created by a data processing application 26 of the computer C3.

Detailed Description Text (117):
A description will now be given, with references to FIGS. 6, 7 and 11, of a marshalling method which makes it possible to publish on the network 4 a data object and its interface.

Detailed Description Text (118):
Non-limitatively, in this example, the marshalling of an object O1 on a computer C1 as illustrated in FIG. 2 is considered.

Detailed Description Text (119):
Naturally, the marshalling can be carried out on any computer C1 to C6 in the network 4.

Detailed Description Text (127):
In the affirmative, a step E5 of marshalling by reference is implemented as described below.

Detailed Description Text (129):
In the affirmative, a step of marshalling by reference E6 is also implemented as described below.

Detailed Description Text (131):
In the affirmative, a third step of marshalling by reference E7 is implemented as described below. Otherwise it is considered that the structure of the data field is simple and a step E8 of marshalling by value is implemented, making it possible to marshall all the data field to the communication language.

Detailed Description Text (132):
The steps E5, E6 and E7 of marshalling by reference constitute a step of

substituting a URI ("Uniform Resource Identifier") computer address for the data field.

Detailed Description Text (133):
This substitution step E5, E6 and E7 consists in reality of not directly marshalling the entire complex data field, but substituting for it a computer address making it possible to find this data field if necessary.

Detailed Description Text (134):
These same steps of marshalling by reference E5, E6 and E7 also comprise a step of storing this URI computer address associated with the data field in tables of associations T and T' as illustrated in FIGS. 6 and 11.

Detailed Description Text (136):
By way of example, here, in the marshalling of the data object O1, two other data objects O2 and O3, included in the data fields of the data object O1, are stored in the first table T in association respectively with a computer address URI1 and URI3.

Detailed Description Text (139):
This marshalling method avoids marshalling all the data fields of a computer object O1.

Detailed Description Text (143):
The marshalling is deferred, in order to be performed only if necessary, when the data field or the interface is requested by another data processing application and must be transferred over the communication network.

Detailed Description Text (146):
For implementing the marshalling method according to the first aspect of the invention, means of reading data fields, substitution and storage are incorporated in the microprocessor 500 of the computer C1, the read only memory 501 storing the program instructions for implementing the method and the random access memory storing, in registers, the variables modified during the execution of the marshalling, and in particular the tables of associations T, T'.

Detailed Description Text (147):
Moreover, the present invention also concerns a method of transferring a data object over a computer communication network as illustrated in FIG. 8. This computer object transfer method makes it possible to call for data objects on the communication network, as soon as these objects have been made visible by the marshalling method as described previously and are referenced in a table of associations T by means of their URI computer address.

Detailed Description Text (167):
In the negative, in a conventional manner on a communication network, the computer sends back, by way of response, an exception, with a message of the type "object absent".

Detailed Description Text (168):
On the other hand, if the object O2 is found in the table of associations T, a marshalling step E16 is implemented to marshall this identified data object O2 to the XML communication language defined by the communication network.

Detailed Description Text (170):
The reception, extraction, identification, marshalling and transfer means of the transfer device are incorporated in the microprocessor 500 of the computer C1, the read only memory 501 storing the program instructions for transferring an object and the random access memory containing registers for storing the variables modified during the execution of the transfer method.

Detailed Description Text (171):
There is thus obtained, by virtue of the marshalling method according to the invention, a distributed objects system on the communication network similar to the Web formed by all the documents accessible on this network.

Detailed Description Text (193):
Preferably, although this is not necessary, a step E23 of marshalling this data object O3 makes it possible to unmarshall the data object O3 in a programming language used by the data processing application used on the distant site.

Detailed Description Text (194):
Naturally, this marshalling step E23 could be deferred in time until the data processing application of the distant site needs this object O3.

Detailed Description Text (202):
Means of receiving a request, extracting an address and an object, associating this address and object in the table T by substitution or addition, and marshalling this object, are incorporated in the microprocessor 500 of the computer C1. The read only memory 501 stores the program instructions adapted to implement the updating method and the random access memory stores the updated table of associations T.

Detailed Description Text (215):
By virtue of the communication language described previously, and the marshalling of a data object, it becomes possible also to execute at a distance, on a computer communication network 4, a function on this data object.

Detailed Description Text (236):
On the other hand, when the interface 11 has been identified, a marshalling step E56 is implemented if necessary in order to marshall this interface to the XML language.

Detailed Description Text (237):
This marshalling step E56 is obviously implemented only if the interface I1 stored in the table of associations T' is in a programming language and not already marshalled.

Detailed Description Text (261):
In the affirmative, a marshalling step E64 is performed in order to marshall these input arguments from the programming language, here C++, to an XML communication language, according to the marshalling method described previously.

Detailed Description Text (272):
Preferably a test step E73 checks whether input arguments are necessary for executing this function. In the affirmative, a step E74 of extracting these input arguments is performed, and then a reverse marshalling step E75 marshals all these input arguments from the XML communication language to a programming language such as C++.

Detailed Description Text (281):
On the other hand, if a result must be supplied, such as a modified object, a marshalling step E85 is then performed on the data object O2 on which the function was performed, in order to marshall this data object O2 from its programming language C++ to the XML communication language.

CLAIMS:

1. A method of transferring a data object on a computer communication network, comprising the steps of: receiving a computer request for a transfer; extracting a computer address from the computer request; identifying a data object associated

with the computer address; marshalling the identified data object to a communication language that conforms with a communication protocol of the computer communication network; and transferring the marshalled data object.

4. A method according to claim 3, further comprising the step of marshalling a data object extracted from the computer request for an update to a programming language used by a computer application of a site in the computer communication network.

8. A method according to claim 1, wherein the marshalling step includes: reading a data field included in the data object; substituting, for the data field, a computer address associated with the data field when a structure of the data field is complex; and storing the computer address associated with the data field in a table of associations.

15. An apparatus for transferring a data object on a computer communication network, comprising: means for receiving a computer request for a transfer; means for extracting a computer address from the computer request; means for identifying a data object associated with the computer address; means for marshalling the identified data object to a communication language that conforms with a communication protocol of the computer communication network; and means for transferring the marshalled data object.

18. An apparatus according to claim 17, further comprising means for marshalling a data object extracted from the computer request for an update to a programming language used by a computer application of a site in the computer communication network.

22. An apparatus according to claim 15, wherein the means for marshalling includes: means for reading a data field included in the data object; means for substituting, for the data field, a computer address associated with the data field when a structure of the data field is complex; and means for storing the computer address associated with the data field in a table of associations.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L3: Entry 21 of 58 File: USPT Nov 4, 2003

DOCUMENT-IDENTIFIER: US 6643652 B2
TITLE: Method and apparatus for managing data exchange among systems in a network

Application Filing Date (1):
20010112

Detailed Description Text (71):
In the preferred embodiment the meta-data store contains the definition of each type of object in the system, its attributes, and some basic properties of those attributes. Further, for each type of object, it contains a reference to the methods to invoke, to insert, update, delete or fetch a given instance of that object from the persistent store.

Detailed Description Text (88):
In a preferred embodiment all business objects that Saba's Application server manipulates are derived from a single base class called SabaObject. The SabaObject class provides save, restore, and delete capabilities by implementing the persistence layer architecture. All subclasses of SabaObject then inherit this behavior and rarely if ever override it. Every SabaObject is expected to know which class it belongs to, and how that class is registered in the meta-data store. Thus each subclass of SabaObject stores a class identifier so that it can tell the system which entry in the meta-data store it corresponds to. Every SabaObject also stores a state flag that determines whether this is a new object, or it is an object that already exists in the data store. This state then determines whether the object invokes an insert method or an update method during a save( ) invocation. Every SabaObject has an unchangeable, unique identifier that identifies that particular object in the persistence store. The uniqueness of this identifier is guaranteed across the entire persistence store regardless of the type of object.

Detailed Description Text (103):
The SabaObject implementation then simply picks up these fields during its normal marshalling and unmarshalling of arguments. Further, the SabaObject also performs the basic checks for nullity as it would normally do.

Detailed Description Text (189):
In the alternative embodiment, to develop an application using the BDK, an object model of the application domain should be first developed, retaining a separation between objects that represent business processes and those that represent business data. The two types of objects, obviously, map to session beans and entity beans in EJB parlance. A controller object, for instance, would indicate a session bean whereas an object that persists its data would indicate an entity bean. An application would typically also include UI components (such as JSP pages or servlets) which would use such business components. Thus, there are two primary roles from an application development standpoint: 1. component developer, and 2. component user.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L3: Entry 22 of 58 File: USPT Oct 14, 2003

DOCUMENT-IDENTIFIER: US 6633878 B1
TITLE: Initializing an ecommerce database framework

Application Filing Date (1):
19990730

Detailed Description Text (146):
This portion of the description describes the appropriate usage of the different types of object creation methods.

Detailed Description Text (161):
As parameters, MTS registered business objects are passed by reference as they use standard marshalling

Detailed Description Text (169):
If the marshalling of data from client to server is done by collection, beware to use the wrapper collection provided on the MTS site. MTS may not work correctly when passing the VB standard collection object. It is known to cause runtime errors.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L3: Entry 28 of 58 File: USPT Nov 26, 2002

DOCUMENT-IDENTIFIER: US 6487607 B1
TITLE: Methods and apparatus for remote method invocation

Application Filing Date (1):
19980320

Brief Summary Text (16):
In object-oriented systems, a "class" provides a template for the creation of "objects" (which represent items or instances manipulated by the system) having characteristics of that class. The term template denotes that the objects (i.e., data items) in each class share certain characteristics or attributes determined by the class. A class thus defines the type of an object. Objects are typically created dynamically during system operation. Methods associated with a class are generally invoked (i.e., caused to operate) on the objects of the same class or subclass.

Brief Summary Text (20):
In order to perform RMI, these machines use code referred to as stubs and skeletons created by an RMI compiler. A stub resides on a client machine and includes a reference to a remote object and acts as a proxy for the remote object. The skeleton is also a proxy for the remote object but it resides on the server machine containing the remote object. Thus, the stub resides on a client machine making a call for invocation of a method of a remote object, and the skeleton resides on a server machine containing the remote object. In addition, remote objects are referenced via interfaces, which are collections of related constants and methods. The stub and skeleton typically must be type-specific based on the type of remote object involved in the call. Therefore, varying types of stubs and skeletons must exist for RMI involving varying types of objects. Stubs and skeletons are explained in for example, the following document, which is incorporated herein by reference: Jamie Jaworski, "Java 1.1 Developer's Guide, Second Edition," pp. 371-383, Sams.net Publishing (1997). They are also explained in the Remote Method Invocation Specification identified above.

Brief Summary Text (21):
Accordingly, a need exists for generic code for use in RMI involving varying types of objects.

Detailed Description Text (5):
The proxy calculates a method hash for the method and transmits a call, possibly via RMI. The call contains, but is not limited to, the following information: an identifier for the object receiving the call; the method hash; and the marshalled parameters (according to the types specified in the method object). The parameter marshalling may be done in a generic proxy's code, since it only depends on knowledge of the argument types in the method object.

Detailed Description Text (31):
RMI 605 returns a response 610 using generic code 607. The generic code is used to do the following: look up the method based on the method identifier; unmarshal the parameters based on their types as indicated in the method object; invoke the

method on a remote object implementation; and marshal the return result(s) (based on the type) to the client. The response may include an identification of the type of object transmitted, the data constituting the state of the object, and a network-accessible location in the form of a URL for code that is associated with the object. The response may be transmitted as a stream.

Detailed Description Text (34):
Further details on the use of a method hash are disclosed in U.S. patent application entitled "Method and System for Deterministic Hashes to Identify Remote Methods," which was previously incorporated herein by reference. Marshalling involves constructing an object from a byte stream including code or a reference to code for use in the construction. Marshalling and unmarshalling are explained in U.S. patent application Ser. No. 08/950,756, filed on Oct. 15, 1997, and entitled "Deferred Reconstruction of Objects and Remote Loading in a Distributed System," now allowed, which is incorporated herein by reference.

Detailed Description Text (35):
Server machine 606 reads the object id and method hash (step 704), and it looks up the method object for the call using the method hash (step 705). Server machine 606 unmarshals the parameters for the operation given the types of the parameters specified in the method object (step 706). Step 706 may involve building a method table, which compiles values for particular methods and is initialized when a remote object is created and exported. The generic code creates a correspondence between a method hash and a particular method object. Thus, by using the hashes in the method table, different skeletons typed to different types of objects is not required for method invocation to correspond method hash to method object.

Previous Doc  Next Doc  Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L3: Entry 30 of 58 File: USPT Sep 3, 2002

DOCUMENT-IDENTIFIER: US 6446070 B1
** See image for Certificate of Correction **
TITLE: Method and apparatus for dynamic distributed computing over a network

Application Filing Date (1):
19980226

Detailed Description Text (5):
Unlike conventional systems, a task in the dynamic distributed system consistent with the present invention can be written once and executed on any server computer in a network. This capability is particularly advantageous in a heterogeneous network because the task does not have to be ported to every platform before it is executed. Instead, a generic compute task designed in accordance with the present invention is loaded on each system. This generic compute task is capable of executing a wide variety of tasks specified by the client at runtime. For example, one can develop a type called "Compute" and a generic compute task which accepts the "Compute" type in an object-oriented language, such as Java. Java is described in many texts, including one that is entitled "The Java Language Specification" by James Gosling, Bill Joy, and Guy Steele, Addison-Wesley (1996), which is hereby incorporated by reference. The client creates a task having a subtype of the type "Compute" and passes an object corresponding to task to the generic compute task on the server. A remote procedure call mechanism downloads the object to the server and the generic compute task which executes the task.

Detailed Description Text (29):
Typically, the client will indicate in the request package where the particular type is located. The skeleton can download the requested type from a object/method repository and can cache the type for future server requests. Also, the requested type could also be located on the client. For example, in Java and RMI the class containing the particular type is located in the given codebase URL (universal record locator) transmitted by the client. Dynamic class loading features in RMI facilitate the automatic downloading of the class using the codebase. These types enable the skeleton to parse the task request and extract the appropriate data and parameters. The steps outlined above make the parameters and data readily available for further processing.

CLAIMS:

1. A method performed on a computer system having a primary storage device, a secondary storage device, a display device, and an input/output mechanism which enables a client to dynamically distribute to a server computer in a collection of server computers a task developed in a programming language compatible with each of the server computers, the method comprising the steps of: selecting a server from a plurality of heterogenous servers to process a task based upon the overall processing load distribution among the collection of server computers and the specialized computing capabilities of each server computer; marshalling parameters and data into a task request which further comprises the substeps of, determining if code and data types related to the requested task are present on the selected server, and downloading the code and related data types onto the selected server

when the code or data types are not present on the selected server; invoking a generic compute method associated with the selected server which executes the task and further comprises the substeps of, providing the task as a parameter to the generic compute method, and indicating to the server that results from a computed task should be stored in a result cache on the selected server for subsequent tasks to use; and receiving the computed task back from the selected server for further processing on the client.

3. A computer readable medium containing instructions for controlling a computer system to perform a method for enabling a client to dynamically distribute to a server in a collection of servers a task developed in a programming language compatible with each of the servers, the method comprising the steps of: selecting a server from a plurality of heterogeneous servers to process a task based upon an overall processing load distribution among the collection of servers and specialized computing capabilities of each server; marshalling parameters and data into a task request which further comprises: determining if code and data types related to the requested task are present on the selected server, and downloading the code and related data types onto the selected server when the code or data types are not present on the selected server; invoking a generic compute method associated with the selected server which executes the task and further comprises: providing the task as a parameter to the generic compute method, and indicating to the server that results from a computed task should be stored in a result cache on the selected server for subsequent tasks to use; and receiving the computed task back from the selected server for further processing on the client.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L3: Entry 33 of 58 File: USPT Jun 25, 2002

DOCUMENT-IDENTIFIER: US 6412010 B1
** See image for Certificate of Correction **
TITLE: APPARATUS AND METHOD FOR IMPLEMENTING A NETWORK PROTOCOL THAT SUPPORTS THE TRANSMISSION OF A VARIABLE NUMBER OF APPLICATION-USABLE OBJECT OVER A NETWORK AS A SINGLE NETWORK TRANSMITTABLE CONTAINER OBJECT AND THE RE-CREATION OF THOSE APPLICATION-USABLE OBJECT THEREFROM

Abstract Text (1):
A method and apparatus for implementing a network protocol to support the transmission of a variable number of application-usable objects as a single network transmittable container object and the re-creation of the application-usable objects therefrom, which includes marshalling application-usable objects into marshalled objects, where each application-usable objects corresponds to a different marshalled object, and each marshalled object comprises frames of data; creating a two-dimensional array to transmit the application-usable objects, where a first dimension represents a given marshalled object by referencing a second dimension array, and a second dimension array comprises frames of data of a given marshalled object; and storing each marshalled object in the two-dimensional array.

Application Filing Date (1):
19990618

Brief Summary Text (11):
Thus, the invention may comprise an apparatus for implementing a network protocol to support the transmission of a variable number of application-usable objects over a network as a single network transmittable container object and the re-creation of said number of application-usable objects therefrom, comprising a number of computer readable media; and computer readable program code stored on the computer readable media comprising program code for marshalling application-usable objects into marshalled objects, wherein each of the application-usable objects corresponds to a different one of the marshalled objects, and each of the marshalled objects comprises at least one frame of data; program code for creating a two-dimensional array, comprising program code for creating a first dimension array comprising a number of elements, wherein each of the elements of the first dimension array represents a given one of the marshalled objects by referencing a second dimension array comprising a number of elements; and program code for creating the second dimension array, wherein the elements of the second dimension array referenced by a given element of the first dimension array comprise frames of data of a given one of the marshalled objects; and program code for storing each of the marshalled objects in the two-dimensional array.

Brief Summary Text (13):
The invention may also comprise a method for implementing a network protocol to support the transmission of a variable number of application-usable objects over a network as a single network transmittable container object and the re-creation of said number of application-usable objects therefrom, comprising marshalling application-usable objects into marshalled objects, wherein each of the application-usable objects corresponds to a different one of the marshalled

Search Forms
Search Results
Help

objects, and each of the marshalled objects comprises at least one frame of data; creating a two-dimensional array, comprising creating a first dimension array comprising elements, wherein each of the elements of the first dimension array represents a given one of the marshalled objects by referencing a second dimension array comprising elements; and creating the second dimension array, wherein the elements of the second dimension array referenced by the given element of the first dimension array comprise frames of data of the given one of the marshalled objects; and storing the given one of the marshalled objects in the two-dimensional array.

Brief Summary Text (15):
The invention may also comprise an apparatus for implementing a network protocol to support the transmission of a variable number of application-usable objects over a network as a single network transmittable container object and the re-creation of said number of application-usable objects therefrom, comprising means for marshalling said variable number of application-usable objects into a variable number of marshalled objects, wherein each of said variable number of application-usable objects corresponds to a different one of said variable number of marshalled objects, and each of said variable number of marshalled objects comprises at least one frame of data; means for creating a two-dimensional array; and means for storing said given one of said variable number of marshalled objects in said two-dimensional array.

Detailed Description Text (11):
The callback function 310 invokes a function to convert any primitive data types into complex data types so that a method Proxy 308 (hereinafter "the Proxy") can serialize the parameters. Complex data types are primitive data types (booleans, bytes, characters, doubles, floats, integers, longs, and shorts) that are wrapped in a Java wrapper (i.e. java.lang.Boolean, java.lang.Byte, java.lang.Character, etc.), which is useful for manipulating primitive data types as objects. This is done by passing the primitive data type into a defined method, which returns a class wrapper for the given data type. Since all objects are subclasses of java.lang.Object, or the Object root class in Java, parameters are placed into an array of Object (Object [ ], to be discussed).

Detailed Description Text (19):
To package multiple parameters in an ObamData object, the Proxy passes in a number of application-usable objects (which in a preferred embodiment is stored in a one-dimensional array of Object, Object[ ], supra) as a parameter to method getObamData (Object object[ ]) (a method which accepts a one dimensional array named object comprising elements of type Object) of class ObamDataHelper. Method getObamData (Object object[ ]) calls a method getBytes of class ObamDataHelper to serialize each element in Object[ ] into a network transmittable object and to place the serialized object in ObamData[x][0], where the first dimension is represented by x for a parameter in relative position x (i.e., if x is 0, then position 0 occurs before position 1, etc.), and the second dimension is represented by 0 which refers to the beginning of the serialized object. The number of elements in the second dimension of a given element in the first dimension is dependent on the size of the serialized object. The following is an example of how getObamData is invoked:

Detailed Description Text (24):
// Create an array of 3 cells of type Object.

Detailed Description Text (31):
Thus, where the first element in array args starts at 0, args[0] represents a method parameter in relative position 0, args[1] represents a method parameter in relative position 1, and so forth. Each element of one-dimensional args is then placed in an element of two-dimensional ObamData: args[x] is inserted into ObamData [x][y], where x is an element number in the first dimension that represents a method parameter in relative position x, y is an element in the second dimension that represents a first byte, y+1 represents a second byte, and so forth. If any of

the parameters is null, then null is stored in the byte array. This is done for all of the elements in the one-dimensional object array of type Object, such that parameters stored in ObamData are in the same order as the parameters appear in the target method. Once all objects in the one-dimensional array have been serialized into network transmittable data and stored in ObamData, ObamData can be transmitted across a network. Since each parameter is represented by one element of ObamData, an arbitrary number of method parameters can be packaged in a way that allows serialization to take place and stored in a single object (ObamData), and each parameter to be individually deserialized into an application-usable object. Without this protocol, there would be no way to distinguish one parameter from another, or to handle an arbitrary number of parameters or arbitrary types in an automated fashion.

Detailed Description Text (32):
The serialized method information (object handle, method name, and method parameters) is transmitted across a network to a method UISendData of class UIServerImpl. (When an object is serialized, some information about its class is serialized with it so that the correct class file can be loaded when the object is deserialized. This information about the class is represented by the java. io. ObjectStreamClass class.) Method UISendData deserializes the object handle back to its original numeric object, and the method name back to its original String object. It then uses an Extractor which accesses ObamData to extract each network transmittable parameter and deserializes each network transmittable parameter back to its original application-usable object by invoking method getObjects(byte obamData[ ][ ]) of class ObamDataHelper. Method getObjects(byte obamData[ ][ ]) accesses a parameter, x, by accessing element [x][0] in ObamData, and placing it into a one-dimensional array of bytes. Method getObjects(byte obamData[ ][ ]) then invokes method getObject(byte byte[ ]) which takes the one-dimensional array of bytes (which represents a single object, or parameter), deserializes the object (through a Java supported method discussed above), and returns the deserialized object of type Object back to method getObjects(byte obamData[ ][ ]). Method getObjects(byte obamData[ ][ ]) then places the object of type Object into Object [x]. Once this is done for each parameter in ObamData, a one-dimensional array of deserialized, application-usable objects of type Object results, where each element in the array represents a single application-usable parameter, and x is the parameter's relative position. Each application-usable parameter can then be accessed from this array. The actual parameter types are extracted from the one-dimensional array of deserialized objects and placed in an array of Class objects, which is a class that supports special objects that represent Java primitive types. The following is an example of how getObjects is invoked:

Detailed Description Text (42):
Although the network protocol described herein is used in conjunction with Java's serialization, it should be understood that the protocol could also apply to network transmission features other than Java's serialization. This network protocol could equally apply to the a generic application of the marshalling and demarshalling of objects where multiple objects need to be transmitted over a network as a single object. For example, if a network transmission feature required that all objects be marshalled (i.e., packaged for transmission over a network) as an array of bits, a two-dimensional array of bits could be created to support the transmission of multiple parameters over the network and the re-creation of those application-usable objects in a manner described herein.

CLAIMS:

1. An apparatus for implementing a network protocol to support the transmission of a variable number of application-usable objects over a network as a single network transmittable container object and the re-creation of said number of application-usable objects therefrom, comprising:

a. a number of computer readable media; and

b. computer readable program code stored on said number of computer readable media, said program code comprising:

i. program code for marshalling said variable number of application-usable objects into a variable number of marshalled objects, wherein each of said variable number of application-usable objects corresponds to a different one of said variable number of marshalled objects, and each of said variable number of marshalled objects comprises at least one frame of data;

ii. program code for creating a two-dimensional array, said program code comprising:

(1) program code for creating a first dimension array comprising a number of elements, wherein each of said number of elements of said first dimension array represents a given one of said variable number of marshalled objects by referencing a second dimension array comprising a number of elements; and

(2) program code for creating said second dimension array, wherein said number of elements of said second dimension array referenced by said given element of said first dimension array comprise said at least one frame of data of a given one of said variable number of marshalled objects; and

iii. program code for storing each of said variable number of marshalled objects in said two-dimensional array.

2. An apparatus as in claim 1, wherein said program code for marshalling said variable number of application-usable objects into a variable number of marshalled objects comprises program code for serializing said variable number of application-usable objects into a variable number of serialized objects, wherein each of said variable number of serialized objects comprises at least one byte.

7. An apparatus as in claim 6, wherein said program code for marshalling said variable number of application-usable objects into a variable number of marshalled objects comprises program code for serializing said variable number of application-usable objects into a variable number of serialized objects, wherein each of said variable number of serialized objects comprises at least one byte.

13. A method for implementing a network protocol to support the transmission of a variable number of application-usable objects over a network as a single network transmittable container object and the re-creation of said number of application-usable objects therefrom, comprising:

a. marshalling said variable number of application-usable objects into a variable number of marshalled objects, wherein each of said variable number of application-usable objects corresponds to a different one of said variable number of marshalled objects, and each of said variable number of marshalled objects comprises at least one frame of data;

b. creating a two-dimensional array, comprising:

i. creating a first dimension array comprising a number of elements, wherein each of said number of elements of said first dimension array represents a given one of said variable number of marshalled objects by referencing a second dimension array comprising a number of elements; and

ii. creating said second dimension array, wherein said number of elements of said second dimension array referenced by said given element of said first dimension array comprise said at least one frame of data of a given one of said variable

number of marshalled objects; and

c. storing said given one of said variable number of marshalled objects in said two-dimensional array.

14. A method as in claim 13, wherein said marshalling said variable number of application-usable objects into a variable number of marshalled objects comprises serializing said variable number of application-usable objects into a variable number of serialized objects, wherein each of said variable number of serialized objects comprises at least one byte.

19. A method as in claim 18, wherein said marshalling said variable number of application-usable objects into a variable number of marshalled objects comprises serializing said variable number of application-usable objects into a variable number of serialized objects, wherein each of said variable number of serialized objects comprises at least one byte.

25. An apparatus for implementing a network protocol to support the transmission of a variable number of application-usable objects over a network as a single network transmittable container object and the re-creation of said number of application-usable objects therefrom, comprising:

a. means for marshalling said variable number of application-usable objects into a variable number of marshalled objects, wherein each of said variable number of application-usable objects corresponds to a different one of said variable number of marshalled objects, and each of said variable number of marshalled objects comprises at least one frame of data;

b. means for creating a two-dimensional array; and

c. means for storing said given one of said variable number of marshalled objects in said two-dimensional array.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L3: Entry 52 of 58 File: USPT Mar 3, 1998

DOCUMENT-IDENTIFIER: US 5724588 A
TITLE: Method and system for network marshalling of interface pointers for remote procedure calls

Application Filing Date (1):
19950607

Brief Summary Text (4):
Computer systems typically have operating systems that support multitasking. A multitasking operating system allows multiple tasks (processes) to be executing concurrently. For example, a database server process may execute concurrently with many client processes, which request services of the database server process. A client process (client) may request services by issuing a remote procedure call (RPC). A remote procedure call allows a server process (server) to invoke a server procedure on behalf of the client. To issue a remote procedure call, the client packages the procedure name and the actual in-parameters for the procedure into an interprocess communications messages and sends the message to the server. The server receives the message, unpackages the procedure name and any actual in-parameters, and invokes the named procedure, passing it the unpackaged in-parameters. When the procedure completes, the server packages any out-parameters into a message and sends the message to the client. The client receives the message and unpackages the out-parameters. The process of packaging parameters is known as marshalling, and the process of unpackaging parameters is known as unmarshalling.

Brief Summary Text (5):
Parameters may be marshalled by storing a copy of the value of the actual parameter in a message. For certain types of parameters, the marshalling may involve more than simply storing a copy of the value. For example, a floating point value may need to be converted from the format of one computer system to the format of another computer system when the processes reside on different computer systems.

Brief Summary Text (6):
The copying of the value of an actual parameter has a couple disadvantages. First, when a copy is passed, changes to the original value are not reflected in the copy. For example, if a parameter representing a time of day value is passed from a client to a server by copying, then the copy that the server receives is not updated as the client updates its time of day value. Second, with certain types of parameters, it may be impractical to make a copy of the value. For example, the overhead of copying a large array may be unacceptable. As discussed in the following, it may also be impractical to make a copy of an object when marshalling the object because the object may be large and include various functions.

Brief Summary Text (27):
It is another object of the present invention to provide a method and system for allowing an object to implement methods for class-specific (custom) marshalling and unmarshalling of pointers to the object in a network environment.

Drawing Description Text (4):
FIG. 3 is a block diagram illustrating the data structures generated and code

loaded during marshalling and unmarshalling.

Drawing Description Text (5):
FIG. 4A through 4C are block diagrams illustrating the marshalling of an interface pointer.

Drawing Description Text (13):
FIG. 12A and 12B are block diagrams illustrating custom marshalling to avoid proxy-to-proxy messages.

Drawing Description Text (14):
FIG. 13 is a block diagram illustrating custom marshalling with shared memory.

Drawing Description Text (15):
FIG. 14 is a diagram illustrating data structures for network marshalling of interface pointers.

Drawing Description Text (16):
FIG. 15 is a diagram illustrating data structures for network marshalling of a sample interface.

Detailed Description Text (2):
The present invention provides a method and system for passing pointers to objects as parameters in a remote procedure call in a computer network environment. In a preferred embodiment, a server process passes a pointer to an interface of an object to a client process. The server marshals a pointer to the interface and sends the marshalled pointer to the client. The client process unmarshals the pointer and accesses the passed object using the unmarshalled pointer. The marshalling and unmarshalling techniques of the present invention load code and generate data structures to support the accessing of the object by the client. In the following, marshalling of an interface pointer within a single node of a network is first described followed by a description of inter-node marshalling of an interface pointer. One skilled in the art would appreciate that the inter-node marshalling techniques can be adapted for intra-node marshalling.

Detailed Description Text (3):
Intra-Node Marshalling

Detailed Description Text (4):
FIG. 3 is a block diagram illustrating the data structures generated and code loaded during marshalling and unmarshalling. The data structures and code include the object 301 and stub object 302 within the server, and proxy object 303 within the client. The proxy 303 and stub 302 are created when a pointer to object 301 is passed to the client. The marshalling process of the server marshals the pointer to an interface of object 301 by loading the code for stub 302, assigning an interprocess communications message address for the stub, storing a pointer to an interface of object 301 within the stub, and packaging the message address of the stub and a class identifier of the proxy into a message. The server then sends the message to the client. When the client receives the message, the client unmarshals the pointer to an interface of object 301 by retrieving the class identifier of the proxy and the stub message address, dynamically loading code to create an instance of the proxy, instantiating proxy 303, and storing the stub message address with the proxy 303. The client then accesses the interface of object 301 through proxy 303.

Detailed Description Text (7):
FIGS. 4A through 4C are block diagrams illustrating the marshalling of an interface pointer. Server 401 contains spreadsheet object 403, cell object 404, and cell object 405. Spreadsheet object 403 has an implementation of the IBasic interface. The method GetCell of the IBasic interface is defined by the following prototype.

Detailed Description Text (11):
FIG. 4B is a block diagram illustrating the marshalling of a cell object to the client. When a client wants to retrieve the formula of cell A1 represented as cell object 404, the client executes the following statements.

Detailed Description Text (16):
The above-described marshalling techniques are referred to as standard marshalling. In a preferred embodiment, the present invention allows an object to specify how pointers to it are to be marshalled in a process referred to a custom marshalling. Each object that implements custom marshalling provides an implementation of a custom marshalling interface called IMarshall. The IMarshall interface provides function members to marshal and unmarshal an interface pointer. When marshalling and unmarshalling a pointer to an interface, the method QueryInterface of the object is invoked to determine whether the object implements the IMarshal interface. If the object implements the IMarshal interface, then the function members of that interface are invoked to marshal and unmarshal the interface pointer. In FIG. 5, the functions UnMarshalInterface (step 505) and MarshalInterface (step 511) preferably determine if the object implements the IMarshal interface and invokes the function members of the IMarshal interface as appropriate.

Detailed Description Text (17):
FIGS. 6 through 10 are flow diagrams of the methods and functions invoked during the marshalling and unmarshalling of a pointer to a cell object. As described below, these methods and functions implement standard marshalling.

Detailed Description Text (20):
This function marshals the designated pointer (pInterface) to an interface for an object into the designated message (pstm). In step 601, the function determines whether the object implements custom marshalling. The function invokes the method QueryInterface of the interface to retrieve a pointer to an IMarshal interface. If the object implements custom marshalling, then a pointer (pMarshal) to the IMarshal interface for the object is returned and the function continues at step 603, else the function continues at step 602. In step 602, the function invokes the function GetSTandardMarshal to retrieve a pointer (pMarshal) to an IMarshal interface with default marshalling methods. In step 603, the function invokes the method IMarshal::GetUnmarshalClass pointed to by the retrieved pointer. The method GetUnmarshalClass returns the class identifier of the class that should be used in the unmarshalling process to instantiate a proxy for the designated interface (iid). In step 604, the function invokes the function Marshal to marshal the unmarshal class identifier to the designated message. In step 605, the function invokes the method IMarshal::MarshalInterface pointed to by the retrieved pointer (pMarshal). The method MarshalInterface marshals the designated interface pointer (pInterface) to the designated message. The method then returns.

Detailed Description Text (36):
A developer of an object can provide an implementation of the IMarshal interface to provide custom marshalling and unmarshalling for the object. The IMarshal interface is defined in the following.

Detailed Description Text (37):
A developer may implement custom marshalling to optimize the marshalling process for both intra-node and inter-node marshalling. FIGS. 12A and 12B are block diagrams illustrating custom marshalling of interface pointers. FIG. 12A shows the effects of standard marshalling. In FIG. 12A, cell object 1201 in process P1 has been marshalled to process P2 as represented by cell stub 1202 and cell proxy 1203. Cell proxy 1203 has been marshalled to process P3 as represented by cell stub 1204 and cell proxy 1205. Cell proxy 1203 was marshalled using the standard marshalling of the type shown in FIG. 7. When process P3 invokes a proxy method of cell proxy

1205, the proxy method marshals the parameters and sends a message to cell stub 1204. Cell stub 1204 receives the message, unmarshals the parameters, and invokes the appropriate proxy method of cell proxy 1203. The proxy method of cell proxy 1203 marshals the parameters and sends a message to cell stub 1202. Cell stub 1202 receives the message, unmarshals the parameters, and invokes the appropriate method of cell object 1201. When the method returns, cell stub 1202 marshals any out-parameters and sends a message to cell proxy 1203. When cell proxy 1203 receives the message, the proxy method unmarshals the parameters and returns to its caller, cell proxy stub 1204. Cell stub 1204 marshals and out-parameters and sends a message to cell proxy 1205. When cell proxy 1205 receives the message, the proxy method unmarshals the parameters and returns to its caller. Thus, whenever process P3 accesses the cell object 1201, the access is routed through process P2.

Detailed Description Text (38):
FIG. 12B shows an optimization resulting from custom marshalling. In FIG. 12B, when process P3 accesses cell object 1201, the access is not routed through process P2 but rather is routed directly to process P1. To achieve this optimization, the IMarshal interface for a cell proxy is implemented with custom marshalling. The method IMarshal::MarshalInterface is implemented as shown by the following pseudocode.

Detailed Description Text (40):
FIG. 13 is a block diagram illustrating custom marshalling with shared memory. In certain situations, an object may store its data members in memory that is shared across processes to avoid the overhead of remote procedure calling to access the data members. In FIG. 13, the data 1302 for cell object 1301 is stored in shared memory. Cell object 1303 is created when a pointer to cell object 1301 is marshalled and sent from process P1 to process P2. The custom methods of the IMarshal interface for a cell object are implemented by the following pseudocode.

Detailed Description Text (41):
The parameter DestContext of the method IMarshal::MarshalInterface indicates whether the data of the cell object is stored in shared memory. If the data is not stored in shared memory, then the equivalent of the standard marshalling is performed. If, however, the data is stored in shared memory, then the address of the shared data is marshalled into the message. The method IMarshal::GetUnmarshalClass determines the unmarshal class based on whether the data is in shared memory. If the data is not in shared memory, then the CellProxy class is the unmarshal class. If the data is in shared memory, then the Cell class is the unmarshal class. The unmarshal class and address of the shared data are sent to process P2. After process P2 instantiates the cell object 1303, the process P2 invokes the custom method IMarshal::UnMarshalInterface, which unmarshals the address of the shared data and initializes the object to point to the data.

Detailed Description Text (42):
Custom marshalling can be used to provide more efficient access to immutable objects. An immutable object is an object whose data members (state) cannot be changed. When an interface pointer to an immutable object is passed from one process to another, custom marshalling can be used to make a copy of the object in the other process. Thus, when the other process accesses the object, it accesses the local copy and no interprocess communication is needed. To allow copying of immutable objects, the methods of the IMarshal interface of the immutable objects are implemented in the following way. In a preferred embodiment, the method IMarshal::GetUnMarshallClass specifies that the unmarshal class is the same class as the class of the immutable object. This ensures that the same type of object is instantiated by the other process. The method IMarshal::MarshalInterface stores the data members of the immutable object into a message for passing to the other process. The method IMarshal::MarshalInterface creates no stub. When the other process receives the message, the other process creates an object of the class of the immutable object. The method IMarshal::UnMarshalInterface then retrieves the

data members from the message and initializes the newly-created object. The other process can then access the immutable object.

Detailed Description Text (43):
Inter-Node Marshalling--Overview

Detailed Description Text (44):
FIG. 14 is a diagram illustrating data structures supporting inter-node marshalling of interface pointers on a computer network. When the client and server execute on different computers (nodes) in a network, then inter-node marshalling is used. As shown in FIG. 14, the object 1410 has interfaces that have been marshalled to the client.

Detailed Description Text (50):
Inter-Node Marshalling--Remote Procedure Call

Detailed Description Text (56):
Inter-Node Marshalling--Marshalling an Interface Pointer

Detailed Description Text (57):
In following the marshalling of an interface pointer across a network is described. The network marshalling is implemented as the standard marshalling. An interface pointer is marshalled by invoking the function MarshalInterface as shown in FIG. 6. The function MarshalInterface retrieves the IMarshal interface (either standard or custom) associated with the interface to be marshalled, marshals an unmarshalling class identifier into a stream, and invokes the method MarshalInterface of the IMarshal interface. In the following, the standard network marshalling of an interface pointer is described. The following pseudocode illustrates the method MarshalInterface of the network implementation of the IMarshal interface. The method MarshalInterface is passed a stream, the interface identifier, and a pointer to the interface to be marshalled. The method MarshalInterface first retrieves the pointer to the IUnknown interface corresponding to the object to be marshalled. When the method QueryInterface of an interface is invoked to return a pointer to the IUnknown interface, the method returns a pointer that uniquely identifies an IUnknown interface for the object. Thus, the method QueryInterface of each interface of an object returns the same value as a pointer to the IUnknown interface. The method MarshalInterface then determines whether the ObjectStubTable contains an entry corresponding to the IUnknown interface of the object to be marshalled. If the ObjectStubTable contains such an entry, then an interface for the object to be marshalled has already been marshalled and the object stub already has been created. Otherwise, the MarshalInterface method creates a object stub by calling the function MakeObjectStub, which is described below. The method MarshalInterface then determines whether the interface identifier of the interface to be marshalled is in the InterfaceStubTable. If the InterfaceStubTable contains such an entry, then the interface has already been marshalled and the interface stub already has been created. Otherwise, the method MarshalInterface creates an interface stub by calling the function MakeInterfaceStub, which is described below. When the method MarshalInterface ensures that both an object stub and interface stub exist, it invokes the method Setup of the IRpcChannelBuffer interface of the stub channel, which allows the stub channel to perform any necessary linking to the interface stub. The method MarshalInterface retrieves the IMarshal interface of the object stub and invokes the method MarshalInterface of the IMarshalInterface.

Detailed Description Text (60):
The following pseudocode illustrates the method MarshalInterface of the IMarshal interfaces of the object stub and the stub channel. The method MarshalInterface of the object stub marshals the object identifier, interface identifier, and channel class identifier into the stream, then forwards the invocation to the method MarshalInterface of the stub channel. The method MarshalInterface of the stub channel marshals a protocol sequence network address, end point of the server, and

a channel identifier into the stream. The protocol sequence, network address, and end point are known as the binding data. The method MarshalInterface of the stub channel then registers the IRpcChannelBuffer with the RPC routine. This registration allows the RPC runtime to locate the appropriate channel when a method of the marshalled interface is invoked. This registration registers the protocols with the RPC runtime and allows the RPC runtime to establish a listener. The marshalling of the class identifier of the stub channel allows the client to select dynamically a proxy channel class that corresponds to the stub channel class. In a preferred embodiment, binding information contains all available protocols that the stub channel supports (e.g., TCP/IP, named pipes, NetBIOS). In this way, the client can select a protocol that is most efficient. For example, if the client happens to be executing on the same node as the server, then the client can select a protocol that is optimized to intra-node remote procedure calls.

Detailed Description Text (61):
Inter-Node Marshalling--Unmarshalling an Interface Pointer

Detailed Description Text (62):
In the following the unmarshalling of an interface pointer is described. An interface is unmarshalled by invoking the function UnMarshalInterface as shown in FIG. 9. The function UnMarshalInterface unmarshals the unmarshal class identifier from the stream, creates an instance of that class, and invokes the method UnMarshalInterface of the IMarshalInterface of that instance. The following pseudocode illustrates the method UnMarshalInterface for standard network marshalling. The method UnMarshalInterface invokes a function to make a proxy object, which in turn invokes a function to make an interface proxy.

CLAIMS:

7. The method of claim 6 including:

when the interface does not have associated custom unmarshalling code, sending to the client an indication of standard marshalling code.

8. The method of claim 6 including:

under control of the server,

invoking a query interface function member of the server implementation of the interface to determine whether the server implementation of the interface has access to a custom marshalling interface; and

when the server implementation of the interface has access, invoking a function member of the custom marshalling interface to provide the indication of the unmarshalling code and the information for use by the unmarshalling code.

19. A method in a computer system for custom marshalling of a pointer to an object from a server to a client, the object having data members and function members comprising:

instantiating the object;

determining whether custom marshalling code is provided for the object;

when it is determined that custom marshalling code is provided, executing custom marshalling code to provide an indication of custom unmarshalling code and information describing the object to the custom unmarshalling code; and

when it is determined that custom marshalling code is not provided, executing standard marshalling code to provide an indication of standard unmarshalling code

and information describing the object to the standard unmarshalling code; and

sending the provided indication and information to the client so that the client can execute the indicated unmarshalling code to provide a custom or standard proxy object through which to request services of the instantiated object.

22. The method of claim 21 including:

when the interface does not have associated custom unmarshalling code, sending to the client an indication of standard marshalling code.

23. A method in a comptuer system for marshalling a pointer to an interface from a first server to a client, the pointer pointing to an interface of the first server that is marshalled from a second server, the interface having a function member providing a behavior, wherein the client, the first server, and the second server each have an implementation of the function member, wherein the implementation of the function member of the first server requests the implementation of the function member of the second server to perform the behavior of the function member, comprising:

under control of the first server, sending to the client an indication of unmarshalling code and information indicating the second server; and

under control of the client,

receiving the indication of the unmarshalling code and the information; and

executing the indicated unmarshalling code to generate a pointer to an interface with a client implementation of the function member that when invoked requests the second server to provide the behavior

whereby when the client implementation of the function member is invoked, then the client requests the second server to perform the behavior without requesting the first server to perform the behavior.

30. The computer-readable medium of claim 29 including:

when the interface does not have associated custom unmarshalling code, sending to the client an indication of standard marshalling code.

31. The computer-readable medium of claim 29 including:

under control of the server,

invoking a query interface function member of the server implementation of the interface to determine whether the server implementation of the interface has access to a custom marshalling interface; and

when the server implementation of the interface has access, invoking a function member of the custom marshalling interface to provide the indication of the unmarshalling code and the information for use by the unmarshalling code.

34. A computer-readable medium containing instructions for causing a computer system to marshal a pointer to an object from a server to a client, the object having data members and function members by:

instantiating the object;

determining whether the custom marshalling code is provided for the object;

when it is determined that custom marshalling code is provided, executing custom marshalling code to provide an indication of custom unmarshalling code and information describing the object to the custom unmarshalling code; and

when it is determined that custom marshalling code is not provided, executing standard marshalling code to provide an indication of standard unmarshalling code and information describing the object to the standard unmarshalling code; and

sending the provided indication and information to the client so that the client can execute the indicated unmarshalling code to provide a custom or standard proxy object through which to request services of the instantiated object.

37. The computer-readable medium of claim 36 including:

when the interface does not have associated custom unmarshalling code, sending to the client an indication of standard marshalling code.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L3: Entry 48 of 58 File: USPT Jul 28, 1998

DOCUMENT-IDENTIFIER: US 5787251 A
TITLE: Method and apparatus for subcontracts in distributed processing systems

Abstract Text (1):
The present invention provides an elegant and simple way to provide mechanisms for invocation of objects by client applications and for argument passing between client applications and object implementations, without the client application or the operating system knowing the details of how these mechanisms work. Moreover, these mechanisms functions in a distributed computer environment with similar ease and efficiency, where client applications may be on one computer node and object implementations on another. The invention includes a new type of object, termed a "spring object," which includes a method table, a subcontract mechanism and a data structure which represents the subcontract's local private state.

Application Filing Date (1):
19961118

Brief Summary Text (14):
When the arguments or results are simple values such as integers or strings, the business of marshaling and unmarshaling is reasonably straightforward. The stubs will normally simply put the literal value of the argument into the network buffer. However, in languages that support either abstract data types or objects, marshalling becomes significantly more complex. One solution is for stubs to marshall the internal data structures of the object and then to unmarshal this data back into a new object. This has several serious deficiencies. First, it is a violation of the "abstraction" principle of object-oriented programming, since stubs have no business knowing about the internals of objects. Second, it requires that the server and the client implementations of the object use the same internal layout for their data structures. Third, it may involve marshalling large amounts of unnecessary data since not all of the internal state of the object may really need to be transmitted to the other machine. An alternative solution is that when an object is marshalled, none of its internal state is transmitted. Instead an identifying token is generated for the object and this token is transmitted. For example in the Eden system, objects are assigned names and when an object is marshalled then its name rather than its actual representation is marshalled. Subsequently when remote machines wish to operate on this object, they must use the name to locate the original site of the object and transmit their invocations to that site. This mechanism is appropriate for heavyweight objects, such as files or databases, but it is often inappropriate for lightweight abstractions, such as an object representing a Cartesian coordinate pair, where it would have been better to marshal the real state of the object. Finally, some object-oriented programming systems provide the means for an object implementation to control how its arguments are marshalled and unmarshalled. For example, in the Argus system object implementors can provide functions to map between their internal representation and a specific, concrete, external representation. The Argus stubs will invoke the appropriate mapping functions when marshalling and unmarshaling objects so that it is the external representation rather than any particular internal representation that is transmitted. These different solutions all either impose a single standard marshalling policy for all objects, or require that individual object implementors

take responsibility for the details of marshalling.

Brief Summary Text (22):
The invention includes a new type of object, termed a "spring object," which includes a method table, a subcontract mechanism and a data structure which represents the subcontract's local private state.

Detailed Description Text (7):
The following disclosure describes solutions to the problems which are encountered by object oriented systems designers when attempting to implement schemes for object invocation and for argument passing in distributed systems wherein the arguments may be objects, in ways which do not lock the object oriented base system into methods which may be difficult to change at a later time. The invention includes a new type of object, termed a "spring object," which includes a method table, a subcontract mechanism and a data structure which represents the subcontract's local private state. A method and an apparatus are disclosed for a subcontract mechanism which is associated with each object. Each subcontract contains a client-side portion and a related server-side portion. Each object type has an associated subcontract. The client-side portion of a subcontract has the ability to generate a new spring object, to delete a spring object, to marshal information about its associated object into a communications buffer, to unmarshal data in a communications buffer which represents its associated object, to transmit a communications buffer to its associated server-side subcontract, which includes either transmitting an object from one location to another or transmitting a copy of an object from one location to another. The server-side portion of the subcontract mechanism includes the ability to create a spring object, to provide support for processing incoming calls and related communications buffers and to provide support for revoking an object.

Detailed Description Text (78):
When some software (typically a stub method) decides to read an object from a communications buffer, it must chose both an initial subcontract and an initial method table based on the expected type of the object. It then invokes the initial subcontract, passing it the initial method table.

Detailed Description Text (87):
The subcontract invoke operation is only invoked after all the argument marshaling has already occurred. In practice it was noted that there are cases where an object's subcontract would like to become involved earlier in the process, so that it can either write some preamble information into the communications buffer or set flags to influence future marshalling.

Detailed Description Text (89):
For example, in some situations a subcontract might use a shared memory region to communicate with a server. In this case when invoke.sub.-- preamble is called the subcontract can adjust the communications buffer to point into the shared memory region so that arguments are directly marshalled into the region, rather than having to be copied there after all marshalling is complete.

CLAIMS:

2. A computer program product for use in a computer system having a memory, a processor, an input/output system, and a mechanism for communicating with distributed computer systems, the computer program product comprising a computer usable medium having computer readable program code mechanisms recorded therein which include:

a first spring object stored in the memory, said first spring object including program logic for executing a call from an application procedure to a first object which comprises a data structure, said first object referring to an object

implementation of said first spring object, without said application knowing a location of said first object or a location of said first object's implementation and without said application knowing details of how arguments must be marshaled for calling said first object;

a stub program mechanism stored in the memory which comprises program logic to assist in executing calls on said first object by marshalling arguments for said first object into a communications buffer;

a plurality of client-side subcontract mechanisms, which comprise program logic to perform remote invocation of operation calls, wherein a spring object within an application can be associated with one of said plurality of client-side subcontract mechanisms; and

wherein a client-side portion of said client-side subcontract mechanism is not required to have knowledge of types of said arguments being sent to said first object; and wherein said client-side portion of a subcontract mechanism has program logic to receive from said stub program mechanism a method identifier and a pointer to a communications buffer which contains arguments to be sent in a call to said first object, wherein said client-side portion of a subcontract mechanism has program logic to add additional data to said communications buffer to provide additional information to said first object and program logic to transmit selectively one of a parameter representing a pointer to said communications buffer and said communications buffer's contents to said first object.

6. A subcontract mechanism for use in a distributed computer system, said computer system having a memory, a processor, an input/output system and a communications system for communicating with other computer systems, said subcontract mechanism comprising:

a first client-side mechanism which resides in a memory and which is distinct from a client-side stub and distinct from an object manager, said first client-side mechanism comprising program logic for executing operations under computer control on an object associated with said subcontract, wherein said object comprises a data structure, and wherein said first client-side mechanism further comprises program logic to execute calls on said object associated with said subcontract by marshalling arguments for said object, including a case wherein one of said arguments is itself an object, into a communications buffer and contains program logic to communicate selectively one of a pointer to said communications buffer and said communications buffer's contents to a second object, and wherein a second client-side mechanism associated with a different subcontract can be called to process said case wherein one of said arguments is itself an object; and

a server-side mechanism which resides in a memory and which is distinct from a server-side stub and distinct from an object manager, said server-side mechanism comprising program logic to associate with said client-side mechanism for exchanging messages with said client-side mechanism and for processing operation calls from said client-side mechanism.

10. In an object oriented system wherein there exists client applications, objects, stubs, object implementations, name registries, program code mechanism libraries, dynamic linker mechanisms and servers, computers containing memory, input/output devices and communications systems connected to other computer systems, a computer implemented method of processing an operation invoked on an object by a client residing in the memory of one of the computers wherein the operation invocation requires marshalling arguments which will permit program logic in an implementation of the object to execute the operation invoked thereon, and wherein one of said arguments may itself be an object, said method comprising the following steps:

receiving an operation invocation on an object in a computer memory by a client-

side stub of said object, said object comprising a data structure;

transforming the operation invocation into an operation call on one of a plurality of subcontracts where different objects within a single application can have different subcontracts associated with said different objects, said transformation comprising the following steps;

determining by the client-side stub whether said operation invocation requires any arguments to be marshalled;

marshaling said arguments into a communications buffer by performing the steps of;

determining by said client-side stub whether any of said arguments to be marshaled is an object;

calling a second subcontract associated with said object which is an argument;

directing said second subcontract associated with said object which is an argument, to marshal said object which is an argument;

receiving said marshaled object from said second subcontract by said client-side stub; and

marshaling any non-object arguments along with said marshaled objects into said communications buffer;

passing a pointer to said communications buffer to a first subcontract which is associated with said object on which said client invoked said operation;

executing said operation call on said one of a plurality of subcontracts; and

returning a result by said one of a plurality of subcontracts to the client-side stub of said object,

whereby an operation invocation on an object which requires marshalling arguments which will permit program logic in an implementation of the object to execute the operation invoked thereon, and wherein one of said arguments may itself be an object, has been efficiently performed.

21. In a distributed computer system having a computer which has a memory, an input/output system, and a communications system for communicating with other computers in the distributed system, and having an object oriented system wherein there exists client applications, objects, stubs, object implementations and servers, a computer implemented method for communicating messages between a client application and a server, wherein the messages include arguments, at least one element of which includes an object, said method comprising the following steps:

executing program logic in a memory for generating an operation invocation on a local object by a client application, said object comprising a data structure;

receiving said operation invocation by a client-side stub of said object;

marshaling arguments associated with said operation invocation into a communications buffer, said marshalling operation comprising the steps of;

determining whether any of the arguments to be marshaled is an object;

for each object which is an argument to be marshaled, identifying one of a plurality of subcontracts which is associated with said object to be marshaled;

using said identified subcontract which is associated with said object to be marśhaled, marshal said object which is being used as an argument to be marshaled into a communications buffer; and

returning a completion signal to said client-side stub;

transforming said operation invocation into an operation call on a client-side subcontract by said client-side stub, said client-side subcontract being one of a plurality of subcontracts;

using said client-side subcontract to transmit said operation invocation and associated communications buffer from said client application to said object's implementation;

using said client-side subcontract to receive a reply from said object's implementation, and to deliver a return communications buffer to said client-side stub;

unmarshalling said results and returned communications buffer by said client-side stub; and

making said results available to said client application.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L3: Entry 15 of 58 File: USPT Jun 22, 2004

DOCUMENT-IDENTIFIER: US 6754885 B1
TITLE: Methods and apparatus for controlling object appearance in a process control configuration system

Application Filing Date (1):
19991123

Drawing Description Text (21):
FIG. 19 depicts an object connection type object model in a system according to the invention;

Drawing Description Text (23):
FIG. 21 depicts a parameter connection type object model in a system according to the invention;

Detailed Description Text (88):
The Object Type class is hierarchical--the branches of the hierarchy represent type categories, with the leaves, or endpoints, of the hierarchy being actual object types with which objects are associated. Instances of Object Types are Parameterized Objects, and may only be directly associated to a single type category (i.e., a specific object type cannot belong to more than one type category). Note, however, that even though an object type can only be directly associated with one type category, it may indirectly be associated with several type categories depending upon where it is in the type hierarchy. Every instance of Object Type has a pointer back to its containing type category, regardless of whether it's acting as a simple object type, or a type category itself.

Detailed Description Text (91):
Since an Object Type which can be referenced by a Typed Object requires a reference to the defining Parameterized Object, only those instances in the Object Type hierarchy be used to serve as the collection point for those same types of objects as they are created. If an Object Type doesn't have a defining reference, is not a container of Typed Objects.

Detailed Description Text (92):
The Object Type class is an abstract class--no instances of Object Type may exist in the database. Subclasses of Object Type are the implementation-standard Object Type class, and the User-Defined Object Type class. The Object Type class contains those methods common between the two subclasses, e.g. methods used to support the hierarchical relationship(s) in the type hierarchy, the containment relationship to Typed Object class, and the reference to its associated definition Type Object instance.

Detailed Description Text (121):
One further restriction: at creation, a Typed Object is prevented from associating with an Object Type (and thereby prevented from being created), unless that Object Type also references an associated defining Parameterized Object which acts as the definition for the Typed Object being created. In an alternate embodiment, when a Typed Object is created and a reference made to its associated Object Type, if that

Object Type doesn't have a reference to the defining Parameterized Object, it simply uses the one from the Typed Object itself.

Detailed Description Text (126):
There may be occasions where it would be desirable to change the type of an object, without having to delete the original object, then create an object of the correct type. One example of where this capability could be useful would be being able to change a station type after a configuration has already been created, and all associations and connections established (this happens often). An alternate embodiment accordingly, permits the type of an object to be dynamically charged.

Detailed Description Text (140):
Additionally, the user picks an already existing object type in the type hierarchy to act as its "template" type, or object type to be used to create from. The user can create a new object type from an existing one in two ways:

Detailed Description Text (190):
1.3.4 Object Connection Type Object Model

Detailed Description Text (219):
There are two relationships that each instance of an Object Connection Type has with the Object Connection Type Specifier class--one is used to specify the source (parent) type, and the other is to specify the sink (child) type. In this way, the Object Connection Type class acts as a join table, relating two object types to determine whether there is a potential connection possible. This class is therefore used as an initial "filter" to determine whether two objects are able to establish a connection before the more complex negotiation between two parameters is allowed to continue.

Detailed Description Text (223):
Each instance derived from an Object Connection Type contains references to two Object Types--one for a Source (Parent) Object Type, the other for a Sink (Child) Object Type. These object types are paired together to determine whether a request to connect two objects together is "legal", or valid, depending upon what types of objects they are.

Detailed Description Text (226):
1.3.5 Parameter Connection Type Object Model

Detailed Description Text (227):
FIG. 21 depicts the classes used in the illustrated embodiment to support connectivity at the parameter level. Note that the class structure presented in FIG. 21 closely parallels that of the object connection type object model presented in FIG. 19.

Detailed Description Text (288):
Appearance Object Model. Objects of the same type appear in a certain way, depending upon which view it's being displayed in. This appearance is defined in an instance of the Appearance Definition class, which describe through the use of macros how a certain type of object appears. The Framework supports both a Implementation-standard, as well as a user-defined, appearance definition of an object type. Finally, a Placeholder Type class links an object type with a view type, with the appearance definition which is dictates how the object type appears on that view type. 1. Placeholder Object Model. This object model details how the placeholder class may actually be abstracted into three different classes: one each to support endpoints, connections, and objects.

Detailed Description Text (301):
Each instance of the View Type class has a one-to-many relationship to instances of the Placeholder Type class. Each view type is capable of displaying one or more

object types, with each valid View Type/Object Type pair represented by an instance of the Placeholder Type class. The appearance of that object on that view type is specified by the associated Appearance Definition object.

Detailed Description Text (329):
Parameterized Object Placeholder objects (from the previous discussion on appearance objects) maintain a reference to their associated Placeholder Type object.

Detailed Description Text (335):
Since a Connection object is a Parameterized Object, it follows that instances of Connection Placeholders (from the previous discussion on appearance objects) maintain a reference to their associated Placeholder Type object.

Detailed Description Text (341):
Unlike instances of the Parameterized Object and Connection Placeholder class, Point Placeholder objects do not contain a reference to a Placeholder Type object, but rather are responsible for determining their appearance using inherited methods and/or data.

Detailed Description Text (371):
There are a number of COM interfaces that are implemented by the IDA application, and managers that are used to synchronize GUI-related and other operations. These are non-automation custom interfaces with associated proxy/stub classes provided by IDA used for marshalling data. The difference between these interfaces and the ones used for external automation is that these are used solely for the coordination of the editors with IDA and are not editor-specific. The automation interfaces are typically unique to the editor they belong to.

Detailed Description Text (449):
In the illustrated embodiment, only one Report Manager can exist in an IDA system, and it is a top-level member of the System Hierarchy. To the user, its representation in the System Hierarchy is an untyped collection, only capable of being opened and closed. It contains three lists, each being a Parameterized Object. Nothing can be added to the Report Manager's "folder" on the System Hierarchy, and none of the three lists can be deleted. The IDA Report Manager relies on a parallel registry of printable views with the following conditions: one or more print views is registered for each Object Type; and of the Object Type's print views is registered as the default.

Detailed Description Text (464):
A Scope Filter Rule acts to populate a Printable Object Collection (POC) by specifying an ordered list of objects that are searched (or whose children are searched) for those matching a specific type. Objects matching this type are added to the POC's temporary list and are further filtered by the POC's Property Filter Rule before being added to the POC's final list. The objects in these lists remain in the order they were added and are subsequently printed in this order by the Report associated with the POC. A Scope Filter Rule can also contain a type of object that is a link to an Active Document object. These objects are treated a little differently by the POC.

Detailed Description Text (665):
A downloadable object which has been checked-in, but not yet downloaded to its target platform is in a state which needs to be made known to the application developer. An object is deemed as being "downloadable" at the time it is created via its association to instances of the Object Type class. A reference to these types of objects is added to the System Workspace object is added at check-in time. Downloadable objects associated with the System Workspace object are removed from the System Workspace once they have been successfully downloaded to their target platform.

Detailed Description Text (740):
Similarly, User objects within a Group object can access many different types of objects.

Detailed Description Text (767):
User objects within a Group may access many different types of objects. Since permissions form a hierarchy, it's possible for a Group object to have multiple permissions within the same Object Type, as well as permission to access different Object Types. This relationship is managed by the class Object Type Permissions.

Detailed Description Text (985):
The following sections describe functions that are implemented by the Control Algorithm Diagram Editor. Most graphical functions apply to all of the visual block/connection type objects which can be configured. Functions specific to the object being edited are in their respective sections.

Detailed Description Text (1478):
Referring to FIG. 114, the Enclosure and Module types available on the Enclosure Element lists are implementation-standard. The user does not have the capability to add or modify these types of objects. Icons for Module types other than FBMs (such as a CP) are provided to enhance the documentation of occupied slots in the Enclosure cells.

Detailed Description Paragraph Table (6):
Parent Child Object Type Capacity Connection Type Object Type Weight HISTORIAN 4000 Historian Connection AIN Block 1 HISTORIAN 4000 Historian Connection PID Block 1

Detailed Description Paragraph Table (8):
Parent Child Object Type Capacity Connection Type Object Type Weight AW70A 2 Serial Connection BW132 1 (NT Station) (Serial Printer) AW70A 2 Serial Connection BW80 1 (NT Station) (Serial Printer)

Detailed Description Paragraph Table (10):
Parent Child Object Type Capacity Connection Type Object Type Weight IE32 4 Nest Connection 1x8CELL 1 (Enclosure) (Cell) 1x8CELL 8 Nest Connection FBM04 1 (Cell) (FBM)

CLAIMS:

1. Apparatus for configuring a control system, the apparatus comprising: a plurality of objects ("configurable" objects) each defining a configurable entity, where each configurable object is associated with a specified object type, one or more objects ("appearance" objects) that identity an appearance of one or more types of configurable objects in one or more views in which those types of configurable objects may be depicted, where each view is associated with a specified view type, a plurality of objects ("placeholder type" objects) that, together, define valid combinations of object types and view types, where each placeholder type object defines an appearance of objects of a specified object type in views of a specified view type in which objects of object type can be displayed, a plurality of persistent documents, each representing a configuration of configurable objects in accord with a selected one of a plurality of views in which configurable objects may be depicted, the persistent document including one or more objects ("placeholder" objects), each placeholder object identifying the location and appearance of a respective configurable object in the selected view to which that persistent document pertains, each placeholder object identifying the appearance object for that respective configurable object identifying the appearance of that configurable object in that selected view, logic that responds to a selected one of the persistent documents by depicting configurable objects whose configuration is represented in that selected persistent document in accord

with locations identified by placeholder objects included in that persistent document and with appearances identified by the respective appearance objects identified by those placeholder objects.

11. Apparatus for configuring a process control system, the apparatus comprising: a plurality of objects ("configurable" objects) each defining a configurable entity in any of (i) a controlled process, (ii) the process control system, (iii) a control level hierarchy, and (iv) the apparatus for configuring the control system, where each configurable object is associated with a specified object type, each configurable object being associated with one or more further objects ("appearance" objects) that identify an appearance of one or more types of configurable object in one or more views in which those types of configurable objects may be depicted, where each view is associated with a specified view type, a plurality of objects ("placeholder type" objects) that, together, define valid combinations of object types and view types, where each placeholder type object defines an appearance of objects of a specified object type in views of a specified view type in which objects of object type can be displayed, each configurable object being associated with one or more still further objects ("placeholder" objects), each identifying the location and appearance of a respective configurable object in a selected one of a plurality of views in which that configurable object may be depicted, each placeholder object identifying the appearance object for the associated configurable object for that selected view, logic that responds to a placeholder object by depicting the associated configurable object in accord with the appearance identified by associated appearance object and in accord with any of the location, size, color and other aspect thereof identified by associated placeholder object.

19. Apparatus for configuring a control system, the apparatus comprising: a plurality of objects ("configurable" objects) each defining a configurable entity, where each configurable object is associated with a specified object type, one or more objects ("appearance" objects) that identity an appearance of one or more types of configurable objects in one or more views in which those types of configurable objects may be depicted, where each view is associated with a specified view type, a plurality of objects ("placeholder type" objects) that, together, define valid combinations of object types and view types, where each placeholder type object defines an appearance of objects of a specified object type in views of a specified view type in which objects of object type can be displayed, a plurality of persistent documents representing a configuration of configurable objects in accord with a one of a plurality of selected views in which configurable objects may be depicted, the persistent document comprising one or more objects ("placeholder" objects), each placeholder object identifying the location and appearance of a respective configurable object in the selected view to which that persistent document pertains, each placeholder object identifying the appearance object for that respective configurable object identifying the appearance of that configurable object in that selected view, one or more connector graphics depicting relationships between configurable objects, logic that responds to a selected one of the persistent document by depicting configurable objects whose configuration is represented in that selected persistent document in accord with locations identified by placeholder objects included in that persistent document and with appearances identified by the respective appearance objects identified by those placeholder objects.

36. A method for configuring a control system, the method comprising: establishing a plurality of objects ("configurable" objects) each defining a configurable entity, where each configurable object is associated with a specified object type, establishing one or more objects ("appearance" objects) that identity an appearance of one or more types of configurable objects in one or more views in which those types of configurable objects may be depicted, where each view is associated with a specified view type, a plurality of objects ("placeholder type" objects) that, together, define valid combinations of object types and view types, where each

placeholder type object defines an appearance of objects of a specified object type in views of a specified view type in which objects of object type can be displayed, defining a plurality of persistent documents, each representing a configuration of configurable objects in accord with a selected one of a plurality of views in which configurable objects may be depicted, the persistent document including one or more objects ("placeholder" objects), each placeholder object identifying the location and appearance of a respective configurable object in the selected view to which that persistent document pertains, each placeholder object identifying the appearance object for that respective configurable object defining the appearance of that configurable object in that selected view, invoking logic that responds to a selected one of the persistent documents by depicting configurable objects whose configuration is represented in that selected persistent document in accord with locations identified by placeholder objects included in that persistent document and with appearances identified by the respective appearance objects identified by those placeholder objects.

46. A method for configuring a process a control system, the method comprising: establishing a plurality of objects ("configurable" objects) each defining a configurable entity in any of (i) a controlled process, (ii) the process control system, (iii) a control level hierarchy, and (iv) the method for configuring the control system, where each configurable object is associated with a specified object type, associating each configurable object with one or more further types of objects ("appearance" objects) that identify an appearance of the associated configurable object in one or more views in which the configurable object may be depicted, where each view is associated with a specified view type, a plurality of objects ("placeholder type" objects) that, together, define valid combinations of object types and view types, where each placeholder type object defines an appearance of objects of a specified object type in views of a specified view type in which objects of object type can be displayed, associated each configurable object with one or more still further objects ("placeholder" objects), each identifying the location and appearance of a respective configurable object in a selected one of a plurality of views in which that configurable object may be depicted, each placeholder object identifying the appearance object for the associated configurable object for that selected view, invoking logic that responds to a placeholder object by depicting the associated configurable object in accord with the appearance identified by associated appearance object and in accord with any of the location, size, color and other aspect thereof identified by associated placeholder object.

54. A method for configuring a control system, the method comprising: establishing a plurality of objects ("configurable" objects) each defining a configurable entity, where each configurable object is associated with a specified object type, establishing one or more objects ("appearance" objects) that identity an appearance of one or more types of configurable objects in one or more views in which those types of configurable objects may be depicted, where each view is associated with a specified view type, a plurality of objects ("placeholder type" objects) that, together, define valid combinations of object types and view types, where each placeholder type object defines an appearance of objects of a specified object type in views of a specified view type in which objects of object type can be displayed, defining a plurality of persistent documents representing a configuration of configurable objects in accord with a one of a plurality of selected views in which configurable objects may be depicted, the persistent document comprising one or more objects ("placeholder" objects), each placeholder object identifying the location and appearance of a respective configurable object in the selected view to which that persistent document pertains, each placeholder object identifying the appearance object for that respective configurable object identifying the appearance of that configurable object in that selected view, one or more connector graphics depicting relationships between configurable objects, invoking logic that responds to a selected one of the persistent documents by depicting configurable objects whose configuration is represented in that selected persistent document in

accord with locations identified by placeholder objects included in that persistent document and with appearances identified by the respective appearance objects identified by those placeholder objects.

Previous Doc Next Doc Go to Doc#